TRABAJO DE FIN DE GRADO

FUNDAMENTOS DE MACHINE LEARNING Y FÍSICA ESTADÍSTICA

Autor:

Ignacio Marchante Hueso

Directores:

Juan José Mazo Torres
David Zueco Lainez

Grado en Física

Facultad de Ciencias

Curso 2019 - 2020

Índice

1. Introducción

En los últimos años, el cada vez más extensivo uso de Inteligencia Artificial en tecnología e informática plantea la cuestión de si la ciencia puede beneficiarse de estos recursos computacionales, de la misma forma que ya se aprovechan herramientas como las simulaciones de Monte Carlo, los algoritmos de resolución de ecuaciones diferenciales, los métodos variacionales o los ajustes numéricos a funciones, por nombrar unos pocos del sinfín de técnicas y métodos útiles.

Machine Learning es un subcampo de la Inteligencia Artificial cuyo propósito es hacer predicciones y estimaciones a través de modelos computacionales que aprenden automáticamente y se refinan a traves de datos. El punto clave es entrenar modelos adecuadamente escogidos con muchos (y buenos) datos, cuestión que con el creciente avance tecnológico y con la emergente popularidad del uso de los datos es más fácil de aprovechar que en los años en los que se originaron estos conceptos. Hoy en día, generar, analizar y almacenar grandes cantidades de datos es posible, lo cual se revierte en buenos modelos capaces de predecir con mayor fidelidad. De hecho, que se generen muchos datos ya solo en el campo de la física (e.g. astronomía, física de partículas, biofísica,ciencia climática) hace que sea valioso contar con herramientas como el Machine Learning que permitan sacarle partido a este recurso.

Lo curioso del Machine Learning es que, a pesar de poder ser útil en el ámbito de la ciencia, no posee "conocimiento físico" de los sistemas que se pretende caracterizar a la hora de hacer sus predicciones. Sin embargo, éstas se acercan a lo que se puede encontrar con otros métodos computacionales que parten de conocer conceptos físicos como el hamiltoniano o la función de partición de un sistema, las ecuaciones del movimiento..., lo cual es sorprendente.

En este trabajo de fin de grado exploraré el uso de la técnicas de Machine Learning y su utilidad como herramienta para un físico enfocándome en su aplicación para la física estadística. Me centraré en un campo más concreto dentro de este: las transiciones de fase en modelos de spines. Estudiaré si mediante Machine Learning se pueden de clasificar adecuadamente configuraciones de un modelo según su fase. A su vez, con esa capacidad de clasificación pretendo estimar los valores críticos de dichas transiciones y comparar con métodos más "estándar" de análisis.

Busco también exponer que las técnicas de Machine Learning funcionan porque están estrechamente relacionadas con la estadística, el ajuste de funciones y con la optimización matemática, de modo que se intentará aclarar su funcionamiento mediante argumentos físicos y matemáticos queriendo disipar la perspectiva inicial de "caja negra" que se puede tener de estas herramientas.

2. Modelos de spines: caracterización "convencional"

A lo largo de este TFG ahondaré en el uso de la técnicas de Machine Learning para caracterizar transiciones de fase en sistemas físicos. Antes de poder explicar los resultados que se pueden obtener empleando Machine Learning, es importante establecer y aclarar correctamente las características relevantes de los modelos físicos que se manejan. Así se tendrá el contexto necesario para ver qué novedad aportan las técnicas modernas que se emplean en este trabajo.

Dentro de la infindad de modelos físicos existentes, acotaré mi trabajo a aquellos centrados en caracterizar las interacciones de colecciones de momentos magnéticos (spines) dispuestos en redes con una cierta geometría. Éstos se conocen como modelos de spines. En mi caso, tomaré el académicamente conocido modelo de Ising y una discretización del modelo XY conocida como *q-state clock model*, y en esta sección los estudiaré empleando cálculo analítico y también simulaciones de Monte Carlo, para poder caracterizar correctamente esas transiciones de fase buscadas.

2.1. Modelo de Ising 2D

Este modelo toma un sistema de spines distribuidos en una red bidimensional, que para este caso se tomará cuadrada, con L nodos por lado. Cada spin ocupa un sitio en la red, de modo que se tiene un total de $N = L \times L$ spines repartidos. Cada uno de ellos tiene sólo dos posibles estados: spin "up" y spin "down". Esto es equivalente a tener una colección de spines $\frac{1}{2}$, como se ve en la Figura 1.

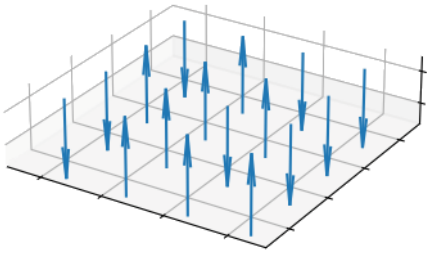


Figura 1: Diagrama de un estado del modelo de Ising. Cada flecha hacia arriba representa un spin "up" y las que apuntan hacia abajo un spin "down".

En las ecuaciones utilizaré que un spin s_i se encuentra en un estado $s_i = \pm 1$, siendo el 1 el estado "up" y -1 el estado "down". Una configuración C es entonces un sistema de N spines distribuidos como se ha descrito con unos ciertos estados $C \equiv (s_1, s_2, ..., s_N)$ que interactúan entre sí a través del siguiente Hamiltoniano:

$$H(C) = -J \sum_{<i,j>} s_i \cdot s_j - h \sum_{i}^{N} s_i \tag{1}$$

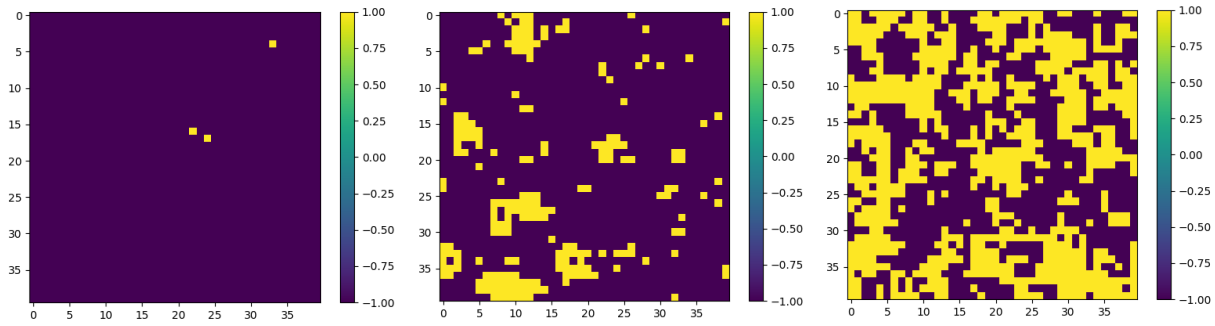
Donde J representa la intensidad de la interacción (que por sencillez tomo unitaria) y h la

intensidad del campo magnético aplicado al sistema. Se impondrá fuera del marco teórico $h = 0$. La primera suma se hace sobre $< i, j >$, es decir, sobre los j primeros vecinos de cada spin i; el de arriba, el de abajo, el de la izquierda y el de la derecha. Lo interesante de este modelo es que para dimensión $d \geq 2$ presenta una transición de fase. La demostración de que para dimensión 2 hay transición de fase se puede encontrar en el Anexo A empleando teoría de campo medio. El valor de la temperatura crítica que se obtiene ($T_c = 4$) no tiene gran acuerdo con lo que se encuentra con simulaciones numéricas, ni con tratamientos exactos más avanzados del modelo, como el que existe en el conocido trabajo de Onsager [1]:

$$\frac{k_B T_c}{J} = \frac{2}{ln(1+\sqrt{2})} \approx 2{,}2691 \tag{2}$$

2.1.1. Simulaciones MC del modelo de Ising

Aparte de los resultados analíticos que se pueden obtener para el modelo de Ising, buscamos también enfoques computaciones para encontrar y caracterizar dicha transición de fase y establecer la temperatura crítica a la que sucede, teniendo un medio alternativo de estudio que pueda mejorar las predicciones de campo medio. Elaboré un programa que hiciera una simulación de Monte Carlo basado en el algoritmo de Metrópolis.Por sencillez, consideré $J = 1$ y las temperaturas están todas expresadas en unidades de k_B. Dicha simulación permite obtener configuraciones propias de una cierta temperatura y mostrarlas por pantalla, como las que se ven en la Figura 2. El programa se llama isingGenerator.py y se puede ver su código en el Anexo D.1.



(a) T=1.0 (b) T=2.26 (c) T=3.0

Figura 2: Representaciones de configuraciones a varias temperaturas para una red de 40×40 spines con condiciones de contorno periódicas.

Se ve a simple vista que existen dos fases diferenciadas: a bajas temperaturas los spines están prácticamente todos en el mismo estado (Figura 2a), y a altas están orientados aleatoriamente (Figura 2c). Hasta ahora la implementación coincide con lo que ya se ha establecido del modelo.

Estas claras diferencias tienen que manifestarse en la evolución con la temperatura de alguna magnitud con la que evaluar el modelo mediante simulaciones. Contamos con la magnetización por spin m empleada en (22) en el Anexo A como parámetro de orden en el desarrolo de campo medio. Defino también la energía por spin e, sencilla de obtener visto (1):

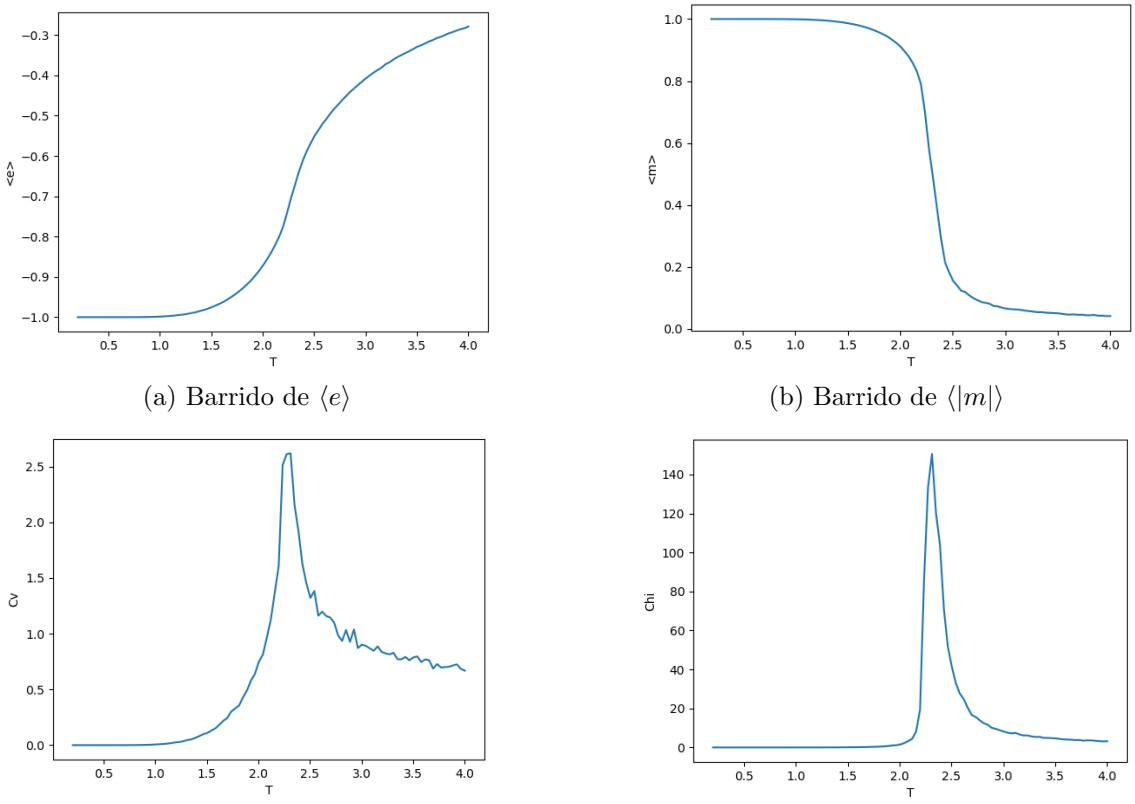
$$e = -\frac{H(C)}{2N} \tag{3}$$

Y con estas dos magnitudes podemos obtener otras dos derivadas de éstas, el Calor específico por spin C_v y la Susceptibilidad magnética χ_s a una cierta temperatura T:

$$C_v = 2N(\langle e^2 \rangle - \langle e \rangle^2) \tag{4}$$

$$\chi_s = N(\langle m^2 \rangle - \langle m \rangle^2) \tag{5}$$

Donde los valores medios surgen de promediar a temperatura fija. Son estas variables de interés las que van a evidenciar la transición que sucede en el modelo de Ising.



(a) Barrido de $\langle e \rangle$

(b) Barrido de $\langle |m| \rangle$

(c) Barrido de C_v

(d) Barrido de χ_s

Figura 3: Evolución con la tempertarura de $\langle e \rangle$, $\langle m \rangle$,C_v y χ_s para el modelo de Ising 2D de tamaño 40×40. Los puntos de las gráficas se obtienen almacenando y operando con 2000 valores de e y m en cada temperatura dejando 10 pasos de MC entre medidas.

Al poder generar configuraciones de spines propias de una temperatura, conseguí efectuar barridos en temperaturas con los que poder evaluar el comportamiento de las magnitudes descritas. El programa que hace esta simulación es IsingBarrido.py, que se encuentra en la Anexo D.2.

La Figura 3 presenta los resultados de la simulación. La curva 3a resulta menos informativa,

simplemente indica que a mayor temperatura los efectos térmicos son más relevantes y la energía del sistema es mayor. Como era de esperar, en la Figura 3b, encontramos una repentina bajada de la magnetización a una temperatura determinada. Es a partir de esa temperatura cuando el orden de largo alcance se pierde y el sistema tiende a estar desordenado. Para valores inferiores de dicha T_c, la magnetización es alta pues la interacción entre spines hace que casi todos lleven la misma dirección. Esta transición de fase se refleja también en C_v y en χ_s (Figuras 3c y 3d), que presentan un pico en la misma zona que la bajada de $\langle m \rangle$. Fijándome en el pico del calor específico obtengo $T_c = 2{,}28$, indudablemente similar a la temperatura crítica obtenida por Onsager [1].

Con estas simulaciones he conseguido caracterizar computacionalmente la transición de fase que sucede en el modelo de Ising. No solo el estudio ha establecido la existencia de una transición de fase de forma cualitativa, sino que puedo llegar a estimar la temperatura crítica.

2.2. *q-state clock model*

El siguiente modelo a considerar añade libertad angular a las posiciones de los spines, permitiendo observar fenómenos diferentes. En esta ocasión se considera una red cuadrada con spines en la que cada uno de ellos está orientado en una cierta dirección coplanar a la superficie de la red (seguiré trabajando en 2D). Cada uno de los spines puede ser por tanto caracterizado por un cierto ángulo (respecto a una referencia común), aunque no se permite cualquier ángulo en el continuo. Dado que el modelo tiene q estados, los posibles ángulos θ permitidos son:

$$\theta = \frac{2\pi}{q} i \ , \quad donde \ \ i = 0, 2, ..., q-1 \tag{6}$$

Ahora la expresión de un estado de un spin i se expresa con un vector bidimensional:

$$\vec{s_i} = (s_x, s_y) = (cos(\theta_i), sin(\theta_i)) \tag{7}$$

Donde he supuesto por sencillez que $|\vec{s_i}| = 1$. De manera análoga al modelo anterior, definimos como configuración C a una colección de N spines como el que acabamos de describir dispuestos en una red; es decir, $C \equiv (\vec{s_1}, \vec{s_2}, ..., \vec{s_N})$.

El Hamiltoniano de una cierta configuración de spines C que determina las interacciones de los elementos de la red sigue la misma filosofía que el del modelo de Ising (1), pero contando con que ahora los spines son vectores bidimensionales:

$$H(C) = -J \sum_{\langle i,j \rangle} \vec{s_i} \cdot \vec{s_j} = -J \sum_{\langle i,j \rangle} cos(\theta_i - \theta_j) \tag{8}$$

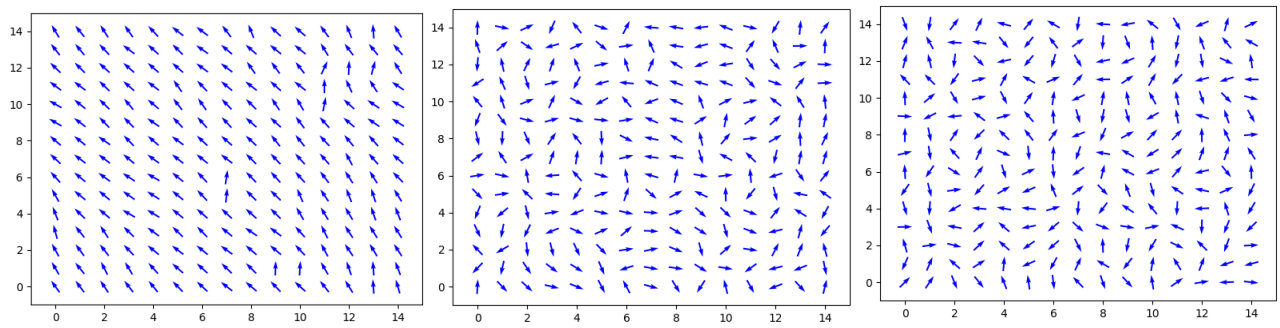
El hecho de que este modelo comparta Hamiltoniano con el modelo XY hace que el *q-state clock model* sea una discretización del mismo, pues cuando $q \to \infty$ los posibles ángulos pasan a pertenecer al continuo y recuperamos el conocido modelo XY. Por otro lado, cuando $q = 2$ el sistema es equivalente al modelo de Ising.

¿Por qué trabajar con el modelo? Porque presenta hasta 3 fases diferentes (el q escogido tiene las tres) y por tanto hasta 2 transiciones de fase diferentes, lo que resulta de interés para poder añadir variedad a los retos a los que enfrentarse con técnicas de Machine Learning. Para el $q = 7$, que usaré, se tiene una fase ordenada tipo ferromagnética, una fase vorticial tipo KT y una desordenada tipo paramagnética, en orden creciente de temperaturas. Se puede encontrar más información sobre el motivo de la existencia de 3 fases en el Anexo B.

2.2.1. Simulaciones MC del *q-state clock model*

Intentaré inferir la existencia de las transiciones de fase esperadas del *q-state clock model* mediante métodos numéricos. Busco de nuevo generar configuraciones del modelo propias de una cierta temperatura, para así hacer barridos en temperaturas con los que caracterizar el modelo y para poder generar además un valioso dataset. Fijé el número de estados a $q = 7$ (recordando que para $q \geq 5$ se tienen dos transiciones de fase), dado que viendo diagramas de fase del modelo en la referencia [3] cpncluí que es un valor para el que la fase BKT está suficientemente extendida en temperaturas y la fase ordenada aún no se ha desvanecido demasiado. Esta relativa holgura en la temperatura es útil a la hora de generar configuraciones de cara a crear un set de datos. Además, cuanto más grande el valor de q más sencillo es ver a simple vista los vórtices y antivórtices.

Comencé creando un programa en Python que realiza simulaciones de Monte Carlo mediante el algoritmo de Metrópolis. El programa se llama ClockStatesGenerator.py y el código se puede consultar en el Anexo D.3. Éste permite generar diagramas como los que se ven en las Figuras 4 y 5.



(a) T=0.3 (b) T=0.55 (c) T=2.0

Figura 4: Representaciones de configuraciones a diferentes temperaturas con q=7 y redes de $15x15$ spines con condiciones de contorno peródicas.

En la Figura 4 se ve cómo las tres fases pueden diferenciarse a simple vista a través de únicamente estas representaciones. La fase BKT es menos clara de establecer "a ojo", motivo de más para seguir simulando y comprobando. Si hacemos q mayor, por ejemplo $q = 30$ la fase BKT se hace más visible sin mayor análisis en la Figura 5. Ver claramente vórtices y antivórtices es más complicado

para q pequeño porque el cambio angular mínimo es considerablemente grande haciendo difícil la visión directa de dichas estructuras.

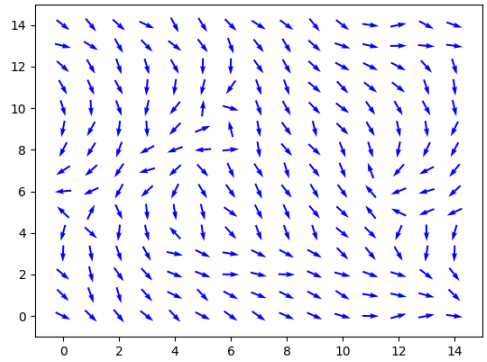


Figura 5: T=0.8 para q=50 en una red de $15x15$ spines

Al querer realizar barridos en temperaturas como antes, el coste computacional de mi implementación demostró ser demasiado grande. El algoritmo de Metrópolis es menos eficiente que otros y el modelo es más complejo y menos optimizable. Recurrí por tanto a simulaciones más rápidas que pudiera encontrar en la red. A través de Mendeley Data descargué un paquete de programas escrito en *'C for CUDA'* basado en el algoritmo tipo clúster de Swendsen-Wang (cambia clústers de spines en cada paso de MC) que además funciona sobre GPU [8]. El programa original realiza los barridos de temperatura, pero evalúa otras magnitudes que no eran de las que buscaba así que hice algún cambio en las mediciones.

Las magnitudes con las que monitorizar las transiciones las consulté en [3]. Comienzo con el módulo magnetización por spin $|m|$, análogo a (22), sólo que ahora hay que tener en cuenta que se trabaja con spines que son vectores de dos componentes:

$$\vec{m} = (m_x, m_y) = \frac{1}{N}(\sum_{i=1}^{N} cos(\theta_i), \sum_{i=1}^{N} sin(\theta_i)) \Rightarrow |m| = \sqrt{m_x^2 + m_y^2} \tag{9}$$

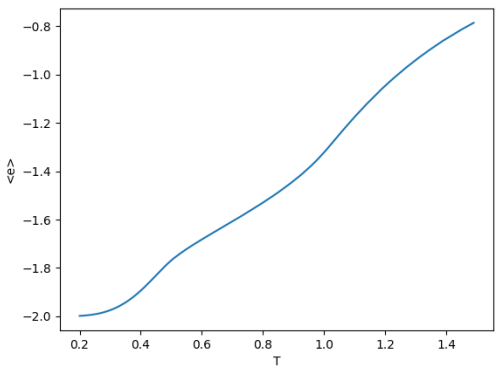
Estoy interesado en el módulo dado que la dirección en la que haya magnetización es irrelevante. Lo interesante es que exista o no magnetización. Continúo con el análogo de (3) para el *q-state clock model*, la energía por sitio de una configuración C, sencilla cuando se conoce (8):

$$e = \frac{H(C)}{N} \tag{10}$$

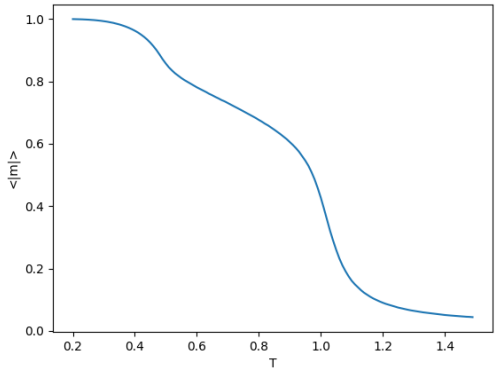
Finalmente el calor específico por spin C_v, que es proporcional a las fluctuaciones térmicas de e:

$$C_v = N\frac{\partial\langle e\rangle}{\partial T} = N\frac{\langle e^2\rangle - \langle e\rangle^2}{T^2} \tag{11}$$

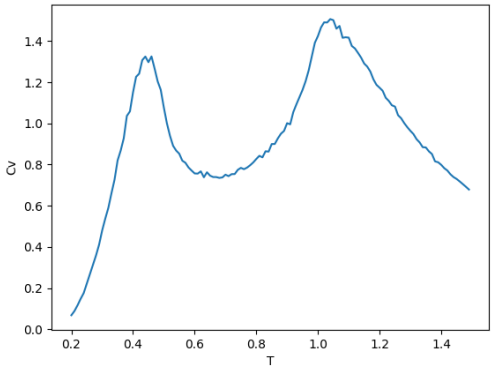
De nuevo, estos los valores medios de las variables en las ecuaciones corresponden a promedios térmicos de los mismos. Puedo entonces realizar barridos en temperaturas como lo hice ya para el modelo de Ising. Las evoluciones térmicas obtenidas se ven en la Figura 6



(a) Barrido en $\langle e \rangle$



(b) Barrido en $\langle |m| \rangle$



(c) Barrido en C_v

Figura 6: Evolución térmica de $\langle e \rangle$, $\langle |m| \rangle$, y C_v para el *7-state clock model* en 2D con tamaño de red de 64×64 y condiciones peródicas de contorno. Promedio 200000 medidas de e y $|m|$, para después computar C_v por cada temperatura simulada.

La Figura 6a presenta un ascenso constante con algunas fluctuaciones. De nuevo, no aporta gran información más allá de comprobar que, en efecto, la energía aumenta con la temperatura, del mismo modo que lo hace la excitación térmica de los spines del sistema. Por el contrario, las Figuras 6b y 6c son más reveladoras. Se ven dos bajadas repentinas en la magnetización y dos picos en el calor específico en las mismas ubicaciones. La temperatura de éstos concuerda con los valores que se pueden encontrar en la literatura al respecto [3] (también realizados con simulaciones MC):

	T_{c1}	T_{c2}
Simulación MC	0.44	1.04
Referencia[3]	0.43	1.03

3. Métodos de Machine Learning

A lo largo de esta sección introduciré los métodos basados en Machine Learning con los que se harán predicciones sobre los modelos trabajados. Las técnicas de Machine Learning pueden parecer una "caja negra"en primera instancia, pero en realidad se asienta sobre la estadistica y la optimización. Esas bases son lo que aclararé a en este apartado.

El Machine Learning es una disciplina con diferentes recursos con funcionalidades y finalidades dispares. Es por ello que se agrupa el aprendizaje ML en tres categorías: aprendizaje supervisado cuando aportamos datos etiquetados (aportando un input con su "solución"), aprendizaje no supervisado cuando los datos se introducen sin etiquetar y aprendizaje por refuerzo, que transmite un objetivo al modelo y ofrece "recompensas" por cumplirlo lo mejor posible, dando libertad al modelo para alcanzarlo. Quiero que los modelos que entrene clasifiquen configuraciones de los modelos de spines descritos anteriormente según su fase, para intentar con ello predecir la temperatura crítica. Es por eso que me centraré únicamente en el uso de técnicas de aprendizaje supervisado que son las idóneas para labores de clasificación.

Esquematizaré el procedimiento de la creación de modelos basados en Machine Learning siguiendo la argumentación que se puede encontrar en [9]. Sea una cierta colección $\mathbf{X}$ de N datos, cada uno con n_f características f, es decir:

$$\mathbf{X} = (\vec{x}_1, \vec{x}_2, ...\vec{x}_N), \;\; donde \;\; \vec{x}_i \equiv (f_1, f_2, ...f_{n_f}) \tag{12}$$

Considero también las correspondientes N etiquetas y_i de cada uno de los datos $\vec{x_i}$ de $\mathbf{X}$ agrupadándolas en $\vec{y}$:

$$\vec{y} = (y_1, y_2, ...y_N) \tag{13}$$

Al conjunto de los datos y sus etiquetas $\mathbf{D} = (\mathbf{X}, \vec{y})$ es lo que llamaré de ahora en adelante *Dataset*. Para conseguir un modelo que haga buenas predicciones debo contar con un dataset $\mathbf{D}$. Dado un cierto dato $\vec{x}$ como los que tenemos en el dataset $\mathbf{D}$, un modelo Machine Learning es una aplicación m que depende de unos ciertos parámetros $\vec{\theta}$ que aplicada sobre $\vec{x}$ nos da una predicción de su etiqueta $\hat{y}$:

$$m(\vec{x}; \vec{\theta}) = \hat{y} \tag{14}$$

He definido el modelo que genera las predicciones que busco pero los valores que han de tener los parámetros del modelo para que las predicciones sean lo mejores posibles no van a surgir de la nada. Necesito contar con otro elemento más: una función que evalúe cuan buenas son las predicciones de m. Para eso se necesitan las etiquetas del dataset, pues evaluaré la efectividad del modelo con una cierta función de coste C que expresa el parecido entre un conjunto de etiquetas $\vec{y}$ de unos ciertos datos $\mathbf{X}$ y las correspondientes predicciones $\vec{\hat{y}}$ que nos da el modelo. Dicho de otra manera, hace falta una función $C(\vec{y}, m(\vec{x}; \vec{\theta}))$ que sea más pequeña cuanto más se asemejen sus dos argumentos.

El objetivo entonces es encontrar el conjunto de parámetros $\vec{\theta}$ del modelo m que minimicen C. Esto habrá de hacerse mediante un método de optimización (algoritmos que modifiquen los parámetros del modelo hacia los óptimos). Para el entrenamiento de los modelos con los que he trabajado he usado algoritmos derivados de uno básico pero efectivo: El método de *Gradient Descent*. Se puede encontrar más información sobre su funcionamiento en el Anexo C.

Lo que se tiene en definitiva ya no es una caja negra, sino un problema estadístico y de optimización. Se necesita un modelo adecuado, un conjunto grande de datos, una función de coste que evalúe la bondad del modelo y un buen método de optimización de la función de coste para tener un modelo capaz de predecir con fidelidad.

Tras definir lo que podrían ser los "ingredientes" de la técnica de ML, ahora explicaré como elaboramos la "receta" para un modelo potente y efectivo. Lo primero es escoger un cierto modelo. Éste puede ser más o menos complejo, y depender de más o menos parámetros. En las siguientes subsecciones se estudiará precisamente que tipos de modelos son útiles para el objetivo perseguido. Una vez escogido el modelo y definida su arquitectura, hay que establecer cúal es la función de coste con la que evaluar las predicciones del modelo y también el método de optimización con el que ajustar sus parámetros. También hay que dividir el dataset en dos particiones o *sets*: una de *entrenamiento* (que tendrá la mayoría de los datos) y otra de *evaluación o testeo*. Se procede a ajustar los párametros del modelo de forma que se minimice la funcion de coste para los datos del *set* de entrenamiento. Una vez hecho esto se evalúa la función de coste con los parámetros ya ajustados en datos no vistos durante el entrenamiento; para eso se tienen los datos del *set* de testeo. Este paso es importante porque por muy bien que ajustemos nuestros datos de entrenamiento los nuevos datos no vistos no tienen en absoluto por que ajustarse tan bien.

Antes de pasar a explicar los modelos de Machine Learning empleados, comentaré una serie fenómenos que requieren atención cuando trabaja uno con estas herramientas.

El primero de ellos es el efecto que tienen el tamaño del dataset y la complejidad del modelo en los resultados que obtenemos. Si el modelo que empleamos es muy sencillo, la falta de riqueza de parámetros puede hacer que no captemos la sutilezas de los datos que queremos analizar. Tendríamos lo que se conoce como *underfitting*. El modelo entrenado no es capaz de comprender las características de los datos y ajustarse a ellos, no ha "aprendido" lo suficiente. Esto también sucede si el dataset es demasiado pequeño.

Por otro lado, si el modelo es muy complejo, es posible que en el proceso de entrenamiento éste pueda ajustarse perfectamente a los datos (sólo a los de entrenamiento), haciendo muy pequeña la función de coste. Pero esa gran fidelidad puede afectar gravemente a la precisión en nuevos datos,al tener un modelo "rígido" demasiado centrado en la información vista durante el aprendizaje. Se tiene lo que se conoce como *overfitting*, y se manifiesta cuando hay altas precisiones en datos de entrenamiento y más bajas en nuevos datos.

El duelo entre emplear modelos sencillos que requieren datasets más pequeños pero que pueden

llegar a presentar *underfitting* o modelos más complicados que necesitan datasets mayores y que pueden presentar *overfitting* es algo que ha de estar siempre en mente cuando se trabaja con Machine Learning para cuestionar cómo mejorar un modelo. Este concepto se conoce como *bias-variance tradeoff*.

Dicho esto pasaré a explicar las arquitecturas de algunos modelos y técnicas, así como establecer su potencial como herramientas de clasificación. Comentaré la técnica de *Logistic Regression* y las bases de funcionamiento de dos tipos de redes neuronales.

3.1. Logistic regression

A pesar de que su propio nombre indica que este es un método de regresión (esto es la predicción de valores y/o funciones continuas), lo cierto es que esta técnica se puede emplear para labores de clasificación binaria (e.g. fase ordenada/desordenada en el modelo de Ising, persona enferma/sana en aplicaciones médicas, señal de partícula supersimétrica/evento de fondo en la búsqueda de partículas supersimétricas [10] [9]). Se fundamenta en emplear una función que, dado un cierto *input* u, exprese la probabilidad de pertenecer a una de las dos categorías de clasificación. Esta función es la función logística o sigmoide:

$$\sigma(u) = \frac{1}{1 + e^{-u}} \tag{15}$$

que al estar acotada entre 0 y 1, nos sirve para este propósito de dar probabilidades:

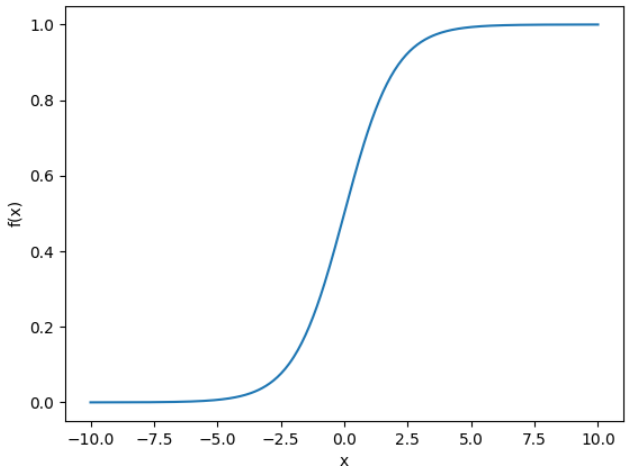


Figura 7: Plot de la función logística

En esta variable u deben estar codificados tanto los parámetros de ajuste del modelo $\vec{\theta}$ como los valores de las variables de entrada de un cierto dato de un dataset $\vec{x} \in D$ con n_f características. Los parámetros serán de la siguiente manera; por cada característica f_i, se emplea un cierto peso w_i multiplicativo y un término independiente b_i. Sumaremos estos n_f términos $u_i = f_i \cdot w_i + b_i$ para computar u:

$$u = \sum_{i=0}^{n_f} u_i = \vec{x} \cdot \vec{w} + \vec{1} \cdot \vec{b} = \vec{\mathrm{x}} \cdot \vec{\theta} \qquad donde \ \mathrm{x_j} \equiv (f_j, 1) \ \ y \ \ \theta_j \equiv (w_j, b_j) \tag{16}$$

En esencia, se está haciendo pasar cada variable de entrada por un filtro lineal y sumando los resultados de cada filtro para evaluar la suma a través de la función logística, que da la probabilidad de que la entrada pertenezca a una cierta categoría. Se considera que si la probabilidad es $> 0{,}5$ el dato de entrada pertenecerá a una categoría y si es $< 0{,}5$ a la otra, logrando una clasificación. Qué variables son más relevantes y merecen mayores pesos o qué términos independientes debe tener cada filtro es lo que se establece durante el entrenamiento.

He descrito el modelo, pero aún queda un punto clave por definir: la función de coste C. Se empleará lo que se conoce como *entropía cruzada*, que es el opuesto de la posibildad logarítmica (que llamaré l), magnitud definida en la siguiente fuente como:

> *"La posibilidad logarítmica es [...] el logartimo natural de la posibilidad. Dada una muestra y una familia paramétrica de distribuciones [...]que podrían haber generado dicha muestra, la posibilidad es una función que asocia a cada parametro la probabilidad [...] de observar dicha muestra."*

Para el caso particular que se trata, la función de coste queda como se indica en [9],cuando contamos con un cierto dataset D como el descrito justo después de la ecuación 13, cuyas etiquetas tienen 2 posibles valores $y_i = 0, 1$:

$$C(\vec{\theta}) = -l(\vec{\theta}) = \sum_{i=1}^{N} -y_i \log \sigma(\vec{\mathrm{x_i}} \cdot \vec{\theta}) - (1 - y_i) \log[1 - \sigma(\vec{\mathrm{x_i}} \cdot \vec{\theta})] \tag{17}$$

Típicamente, se añaden términos de regularización (implementados ya en las librerías de ML) que son simplemente penalizaciones (dependientes de un parámetro que podemos afinar) a la función de coste que pueden ayudar a reducir el *overfitting*, al evitar que durante el ajuste de los parámetros en el entrenamiento éstos den demasiada importancia a los datos en sí.

3.2. Redes neuronales

Esta técnica de Machine Learning supone un paso más allá en complejidad de los modelos empleados. Como su nombre indica, están fundamentados en el uso de neuronas interconectadas formando una red. La red consiste en capas "apiladas" que contienen neuronas , de modo que las neuronas de una capa se conectan con las de la siguiente. Cada una de estas neuronas realiza una serie de operaciones con los *inputs* escalares que recibe y el resultado de dicha operaciones es una respuesta escalar. En ese sentido, una neurona realiza una función similar a la que tiene un modelo de *Logistic Regression*. Hay varios tipos de redes neuronales, aunque en esta memoria me centraré sólo en dos: las Redes Neuronales Profundas (que llamaré en adelante DNN por *Deep Neural Networks*) y las Redes Neuronales Convolucionales (que llamaré en adelante CNN por *Convolutional Neural Networks*). En esta parte introductoria a las redes neuronales comentaré las características comunes a las mismas, para entrar en los detalles de sus arquitecturas y sus peculiaridades en sus

propias secciones.

Ahora entraré a explicar qué hace una neurona y cómo son de manera general las estructuras y conexiones neuronales de las redes. Una neurona es la unidad básica de una red. Sea una cierta neurona de una red . Supongo que la neurona recibe n inputs, que agrupamos en $\vec{x} = (x_1, x_2, ..., x_n)$. La neurona multiplica cada input x_i por un cierto peso w_i (tenemos por tanto n pesos y $\vec{w} = (w_1, w_2, ..., w_n)$) y los suma todos, añadiendo un término independiente extra b. La primera operación que hace la neurona es pues:

$$u = b + \sum_{i=0}^{n} x_i \cdot w_i = \vec{x} \cdot \vec{w} + b \tag{18}$$

La neurona aplica entonces una operación no lineal sobre dicha u. Existen múltiples funciones que son útiles y efectivas para entrenar. Se busca que representen un cambio entre inputs negativos y positivos, para poder representar que la neurona se activa o desactiva. Una de las más extendidas es la *Rectified Linear Unit* (ReLU):

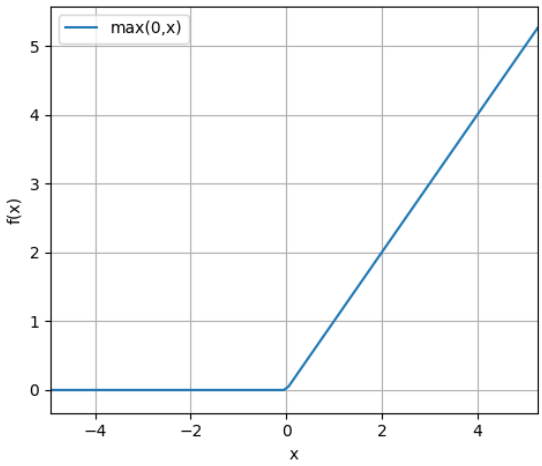


Figura 8: Represntación gráfica de una ReLu

Una vez descrito como funciona una neurona se puede tratar cómo se conectan. Como se ve en la figura 9, las neuronas se organizan en distintas capas, conectando cada capa con la siguiente. Cada una de las flechas corresponde a una conexión que tendrá asociada su propio peso w y cada neurona, un *bias* b como se ha descrito anteriormente. Hay una capa correspondiente a los datos de entrada (la roja), que tendrá tantas neuronas como características posean los datos de entrada. Por otro lado, una red neuronal debe tener una o más capas "ocultas" (en azul). Éstas sirven para procesar la señal de entrada. Finalmente, tenemos la capa de salida (en verde), que tendrá tantas neuronas como categorías de clasificación haya. A cada una de esas neuronas de salida le corresponderá una categoría, con el objetivo de un input active en la salida la neurona de la categoría correcta. En ocasiones, puede ser útil emplear a la salida neuronas activadas por *Softmax*, una generalización de la función logística de *Logistic Regression* a mas de dos categorías. Este tipo de salida, por tanto,

nos da una predicción de la probabilidad con la que la red atribuye cada categoría a un cierto dato de entrada. En cuanto al resto de capas, las neuronas suelen emplear como filtro no lineal ReLUs.

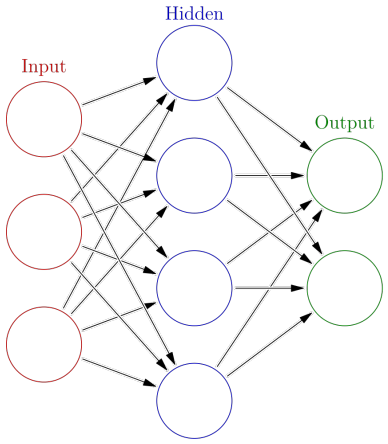


Figura 9: Diagrama de una red neuronal (Creado por Glosser.ca,Licencia CC 3.0,fuente)

A continuación, comentaré las sutilezas de los dos tipos de redes mencionados al inicio de la sección, viendo como se basan en este esquema general para su funcionamiento.

3.2.1. *Deep Neural Networks* (DNN)

Las DNN son bastante similares al esquema descrito anteriormente y aportan pocas novedades al esquema básico. Tienen la limitación de que las capas de neuronas han de ser 1D (no se ha especificado nada al respecto en la descripción general) y se las llama redes profundas porque pueden llegar a tener muchas capas ocultas. Esta gran posible profundidad, hace que puedan llegar a ser modelos muy complejos con capacidad de aprender características avanzadas de los datos.

Más allá de estos detalles, continúo la sección tratando cómo se entrena este tipo de redes. De nuevo, hace falta minimizar una función de coste y emplear un método de optimización. En redes neuronales las etiquetas se dan de forma categórica, como indica la ecuación (19).

$$y = (y_0, y_1, ..., y_{n-1}) \quad donde \;\; y_j = \begin{cases} 1 \quad si \quad i = j \\ \\ 0 \quad si \quad i \neq j \end{cases} \tag{19}$$

De forma que la función de coste (que sigue siendo la entropía cruzada de (17) pero en su versión categórica) queda para un conjunto de N elementos de un dataset D y n categorías como se puede encontrar en [9]:

$$C(\theta) = -\sum_{i=1}^{N}\sum_{c=0}^{n-1} y_c^{(i)} \log \hat{y}_c^{(i)}(\theta) + (1 - y_c^{(i)}) \log[1 - \hat{y}_c^{(i)}(\theta)] \tag{20}$$

Donde θ es el conjunto de parámetros de la red, $y_c^{(i)}$ es la componente c de la etiqueta categórica i del dataset y $\hat{y}_c^{(i)}$, la componente c de la etiqueta categórica predicha por el modelo. Los algoritmos empleados para minimizar la función de coste son por excelencia SGD y Adam, aunque implementados de una manera que haga eficiente el cálculo de gradientes en redes (usan el método de *Backpropagation*).

El entrenamiento de las DNNs es menos directo que el descrito en *Logistic Regression*. Las DNNs se entrenan en uno o varios *epochs*. En esencia, lo que se hace es estimar el gradiente de los parámetros del modelo empleando unos pocos datos (cada conjunto de esos pocos datos se llama *batch*), de manera que se actualizan los valores cada vez que se "recorren" los datos de un *batch*. Cuando se han visitado todos los datos de entrenamiento tomados de *batch* en *batch*, se dice que se ha completado un *epoch*. De un *epoch* a otro los datos que hay en cada *batch* se toman al azar, haciendo que entrenar en más de un *epoch* sea útil para refinar el modelo.

3.2.2. *Convolutional Neural Networks* (CNN)

La necesidad de emplear CNN's surge de la necesidad de reconocer aspectos de datos que no siempre están en una misma localización, por lo que están pensadas para detectar características de datos introducidos con más de una dimensión (Por sencillez y porque sólo trabajaré con inputs bidimensionales, me quedaré en 2D). La mejor forma de entender la utilidad y necesidad de este tipo de redes es pensando en determinar la presencia o ausencia de un objeto en una serie de imágenes; el objeto puede estar en cada imagen con tamaños, orientaciones y ubicaciones diferentes. Las DNNs comentadas reciben sus inputs como una ristra de datos 1D, sin saber la relación entre los datos de entrada, por lo que detectar rasgos con variancia translacional está fuera de sus posibilidades. Es aquí donde las CNNs cobran importancia, al superar las limitaciones que se encuentran en las DNNs.

La diferencia fundamental entre una DNN y una CNN, más allá de la dimensionalidad de los datos de entrada, es la arquitectura del modelo. Si en una capa de una DNN tenemos n neuronas, el equivalente a eso en una CNN es desglosar cada neurona de la capa de la DNN en un filtro de tamaño $H \times L$ neuronas con mismos pesos y bias, teniendo entonces n flitros como el descrito y por tanto una profundidad $D = n$. Además, hay dos tipos de capas, las convolucionales y las de *pooling*. Explicaré qué hacen dichas capas, recomendando fijarse en la Figura 10. Introducimos como input a una CNN una imagen de tamaño $L \times H$ con 1 sólo canal de color y ese input llega a una capa convolucional de misma altura y anchura que la imagen. Cada neurona de cada filtro (tomando profundidad de capa D) hará una operación sobre unos pocos píxeles dando un valor escalar tras su operación. Tendremos una respuesta con escalares de tamaño $L \times H \times D$. Esta operación de filtrado es lo que llamamos convolución, y se aplica igual al input de la red que a otra capa. El otro tipo de capa es la que realiza *pooling*. Este tipo de capas reducen la altura y anchura de los filtros, agrupando los valores de un conjunto de neuronas en uno sólo, quedándose con el máximo valor. Esta operación no afecta a la profundidad de la capa, pues esto se hace en cada uno de los filtros de la misma.

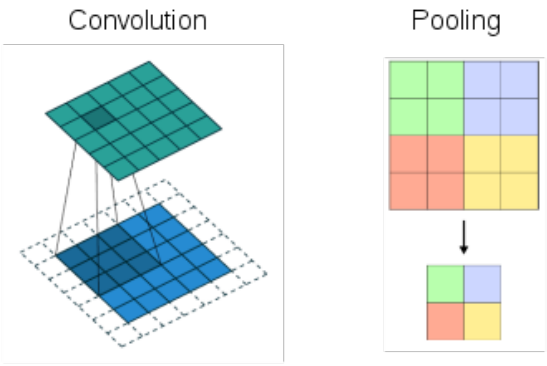


Figura 10: Diagramas del funcionamiento de la convolución y *pooling*. (Licencia CC 4.0,fotografía modificada,fuente)

Explicada la función de las capas se puede estudiar como es la arquitectura básica de una CNN. Como se puede ver en la Figura 11, Se itera esta estructura de convolución+*pooling* (no harán falta más de dos etapas de este tipo en los propósitos que manejaré) y finalmente se introduce lo filtrado en una serie de capas con neuronas en 1D, típicamente con una capa final tipo *Softmax* con tantas neuronas como categorías existentes. Este último paso es como pegarle al final de la CNN una DNN que actúe como clasificador final de lo filtrado de la imagen inicial.

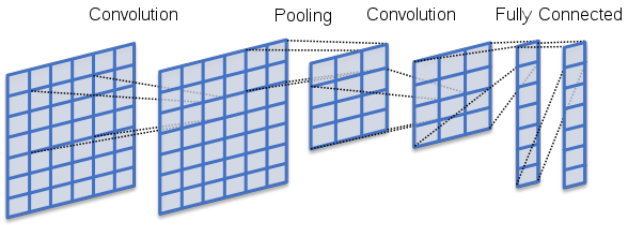


Figura 11: Esquema de la arquitectura básica de una CNN (Licencia CC 4.0,fotografía modificada,fuente)

Más allá de las diferencias de arquitectura, la difrencia en el entrenamiento es escasa; la función de coste es la misma que en las DNN's y los métodos y sutilezas del entrenamiento también lo son. Se añade un elemento al entrenamiento que se puede usar en redes neuronales en general: el set de validación. Es una nueva partición del dataset que se usa durante el aprendizaje para evaluar la bondad del modelo a medio entrenar. De esta manera se puede saber si se están ajustando correctamente los párametros y su rendimiento en datos nuevos.

Con esto quedan explicados los modelos y conceptos de Machine Learning que se han manejado durante la experimentación computacional. Puedo pasar pues a comentar las implementaciones de la técnica a los modelos de spines descritos en las sección anterior y los resultados obtenidos.

4. Resultados

En esta sección se mostrarán los resultados que he obtenido al aplicar diferentes técnicas de Machine Learning al modelo de Ising y al *q-state clock model*. El objetivo es, como se ha comentado a lo largo del documento, intentar establecer las temperaturas críticas de los modelos. Para ello, antes se ha de conseguir un modelo Machine Learning que sepa clasificar adecuadamente una configuración dada en su fase correspondiente. Usaré DNNs y el método de *Logistic Regression* para el modelo de Ising y CNNs para el *q-state clock model* cuando $q = 7$.

4.1. Los Datasets

Además de escoger modelos adecuados para las tareas de clasificación buscadas, hacen falta datasets para poder entrenar modelos. Los datasets que se han empleado han sido creados por mí aprovechando los programas empleados en la caracterización de los modelos. Los programas empleados para generar los datasets son StatesGeneratorPro.py para el modelo de Ising que se pueden consultar en el Anexo D.4 y una modificación del código de [8] que ya no hace ni barridos ni mediciones, sólo guarda configuraciones. En total generé 4: dos del modelo de Ising y dos del *q-state clock model*. Cada modelo de spines tiene un dataset para entrenar modelos y otro auxiliar para las simulaciones con las que se estima T_c). Se pueden ver sus características en la siguiente tabla:

Dataset	**Configuraciones**	**Config./T**	**Rango de T**	**Spines/Config.**
Ising	390000	10000	0,2 $\sim$ 4.0, salto de 0,1	40×40
Ising Aux	128000	6000	0,2 $\sim$ 4.0, 23 valores	40×40
Clock Model	290000	10000	0,1 $\sim$ 1.5, salto de 0,05	64×64
Clock Model Aux	282000	2000	0,1 $\sim$ 1.5, salto de 0,01	64×64

Como anotación, todas las configuraciones han sido generadas tomando $J = 1$ t las temperaturas están en unidades de k_B. *Config./T* representa configuraciones por cada valor de temperatura. El dataset para entrenar el *q-state clock model* tenía el doble de configuraciones originalmente, pero el tamaño del mismo daba errores de memoria en los programas.

4.2. Modelo de Ising: *Logistic Regression*

Comencé empleando la técnica de *Logistic Regression* como primera toma de contacto con los modelos Machine Learning. Tomé el 6º *notebook* de *Jupyter* de esta web. Los *notebooks* de la web son parte del material de apoyo de la referencia [9] y algunos de ellos están dedicados precisamente a clasificar fases del modelo de ising.

Modifiqué el código para que empleara el dataset propio y cambié la proporción de datos de entrenamiento y testeo al 80 %/20 %. El código en cuestión entrena un modelo de *Logistic Regression* que tiene como entrada los estados de los espines de una configuración del modelo de Ising (en forma de array 1D) y que en su salida puede expresar dos etiquetas: 1 para estados ordenados y 0 para desordenados.

Además, se entrena varias veces el modelo cambiando el parámetro λ que controla la intensidad de la regularización y los optimizadores, a fin de hacer comparaciones en el desempeño con los parámetros. Se implementan dos métodos de optimización, el que viene por defecto en el la librería *sklearn* de Python y un SGD. Se emplean para entrenar y testear configuraciones alejadas de la región crítica, pero después de estos pasos se evalúa la precisión en configuraciones de la zona crítica. Comentado esto, la Figura 12 muestra las precisiones obtenidas en los sets de entrenamiento, testeo y críticos para los dos optimizadores según la intensidad de la regularización.

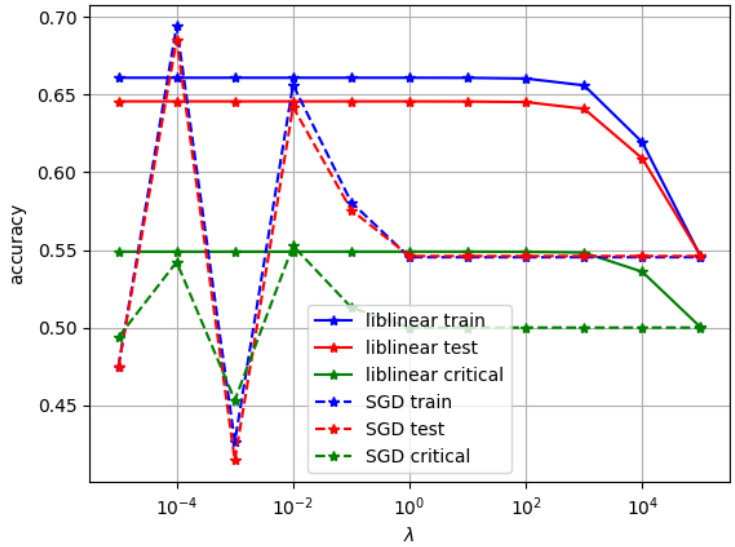


Figura 12: Precisiones obtenidas en los sets de entrenamiento, testeo y crítico para los optimizadores SGD y liblinear según la intensidad de la regularización

Se puede comprobar que en torno a $\lambda = 10^{-2}$ SGD tiene un pico en sus precisiones, aunque liblinear demuestra ser más estable tanto en términos de λ como en términos de repetitibilidad. El pico de $\lambda = 10^{-4}$ no lo considero porque tiene menor precisión en datos críticos, que es donde queremos que el modelo trabaje mejor. Hasta aquí los resultados son una reproducción de lo que se puede encontrar en la sección 7.3.1 de [9].

Tomé como modelo final uno basado en SGD usando $\lambda = 10^{-2}$ que salió con precisiones algo mejores que liblinear para los datos críticos (57 % de acierto), incluyendo ahora los datos críticos en entrenamiento y testeo. Para ello usé una modificación del código original del *notebook* de [9] que entrena sólo para un valor de λ y guarda el modelo en un fichero. Guardé los pesos en un archivo e hice una simulación para tratar de estimar T_c a pesar de las bajas precisiones del modelo entrenado. El programa, llamado Experimento626LR.py se puede encontrar en el Anexo D.5

La simulación hace un barrido de temperaturas sobre el dataset auxiliar. Lo que se espera es que para las temperaturas frías, el promedio de la predicción $\langle y \rangle$ sea cercano a 1 y para calientes cercano a 0. En cuanto a la zona crítica, el $\langle y \rangle$ debería descender desde esos valores altos a los bajos, siendo la temperatura crítica aquella en la que los datos no se pueden clasificar ni como ordenados

ni como desordenados; es decir, la que tenga una $\langle y \rangle$ de 0.5. Lo obtenido se puede encontrar en la Figura 13.

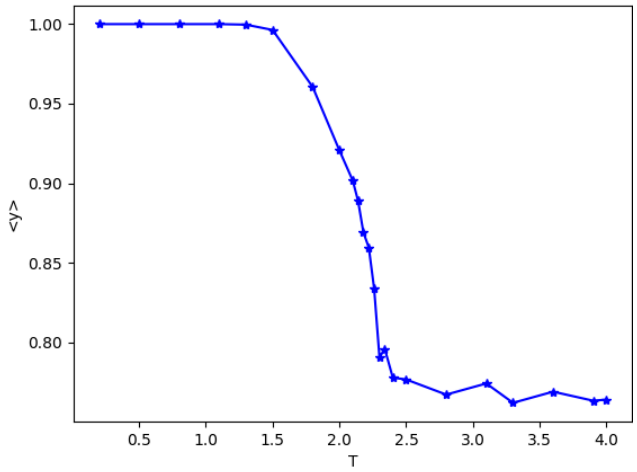


Figura 13: Valor medio de la salida del modelo de *Logistic Regression* a diferentes temperaturas. Por cada temperatura se promedia la etiqueta predicha por modelo para 6000 configuraciones.

El resultado desde luego no es satisfactorio, ya que el modelo etiqueta erróneamente la mayoría de configuraciones claramente desordenadas como ordenadas (aunque clasifique muy bien las ordenadas). Esto sugiere que el modelo empleado es insuficientemente complejo como para aprender las sutilezas de las configuraciones. Es algo que podía verse venir, dado que las precisiones de los modelos no eran especialmente altas en ningún momento, y viendo esta gráfica se entiende exactamente por qué. A pesar del desastroso resultado podemos estimar la temperatura crítica del modelo a través del valor intermedio de $\langle y \rangle$, que ya no es el esperado 0.5 sino 0.88, obteninedo $T_c = 2{,}15$, que por lo menos se acerca al valor conseguido por Onsager.

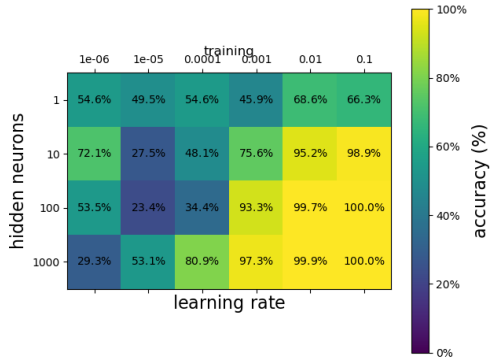
4.3. Modelo de Ising: DNN

Dado que el modelo de *Logistic Regression* ha demostrado ser insuficientemente complejo, cambiar de modelo es la mejor opción. El siguiente paso disponible es el uso de DNNs. Esta vez empleé el 12º *notebook* de *Jupyter* de la misma web de antes [9]. Simplemente modifiqué el código para que trabajara con mi dataset y una salida *Softmax*.

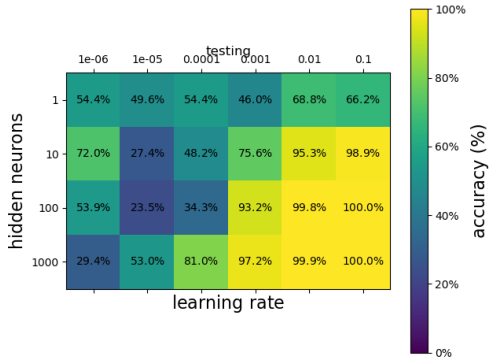
El código entrena una DNN con una sola capa oculta de N neuronas (es una variable). La entrada tiene 1600 neuronas correspondientes al array 1D de las características de una imagen 40×40 y la salida 2, la neurona correspondiente a estar un estado ordenado y la correspondiente a un estado desordenado. Se emplea un método de optimización SGD que depende del correspondiente parámetro de lr (otra variable). A su vez, un entrenamiento recorre 100 epochs con un tamaño de batch de 100 configuraciones.

El código no solo entrena una DNN, sino que hace una *grid search* probando distintas combinaciones de N y lr comprobando la precisión en los datos de entrenamiento, test y críticos (como en las otras implementaciones, no se entrena con datos críticos, pero se separan para testear aparte).

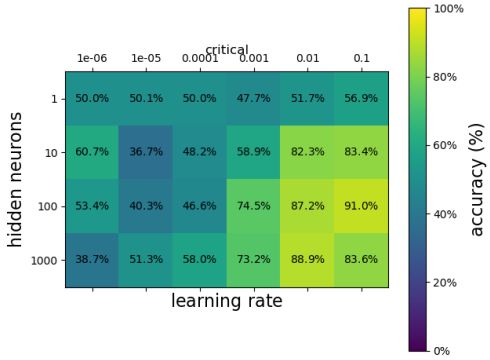
Lo que se obtiene de la ejecución del codigo empleando el dataset creado por mí son los plots de la Figura 14.



(a) Precisiones en datos de entrenamiento



(b) Precisiones en datos de testeo



(c) Precisiones en datos de la región crítica

Figura 14: Grid search de las precisiones de la DNN para el modelo de Ising

El learning rate no llega a ser tan grande como para ser contraproducente de manera que mayores valores suponen mejores predicciones, y generalmente, lo mismo sucede con el número de neuronas. Se busca máxima precisión en datos críticos, que son los que presentan más fluctuación en la precisión de una a otra ejecución. Después de algunas ejecuciones llegué a la conclusión de que la mejor combinación de parámetros era 100 neuronas en la capa oculta y $lr = 0{,}1$. Presenta en general la máxima precisión crítica (muy cercana al 90 %) y la presenta en muchas de las ejecuciones.

Con lo hecho hasta aquí he reproducido los resultados a los que se llega en la sección 9.5.4 de la referencia [9]. Conseguido esto guardé un modelo con los parámetros que he mencionado mediante otra modificación del código del *notebook* ya mencionado que hace un solo entrenamiento y guarda el modelo resultante. Gracias a éste, pude realizar una simulación con la que estimar T_c similar a la que produjo la Figura 13 para *Logistic Regression*. Mismo dataset auxiliar y mismo concepto: evaluar la predicción del modelo sobre esos nuevos datos para encontrar la temperatura para la que el modelo prediga cada fase al 50 %, esperando que para temperaturas bajas la respuesta corresponda a estados ordenados y a altas, desordenados. Cuento ahora con un elemento diferente, y es que la salida no indica la etiqueta predicha, sino la probabilidad de escoger cada posible etiqueta. Esto hará más

clara la ubicación de la temperatura crítica y es una magnitud más informativa. La simulación en cuestión la hace el programa Experimento626DNN.py (Código en el Anexo D.6) y el resultado se ve en la Figura 15.

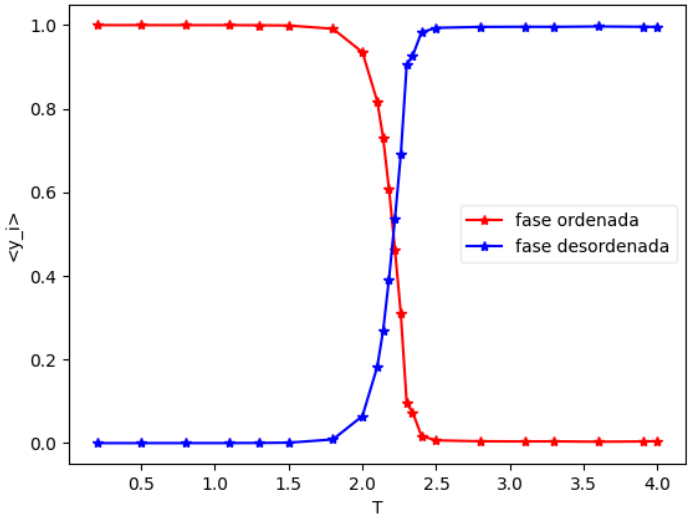


Figura 15: Evolución del valor medio de la salida de la DNN con la temperatura. Por cada temperatura se promedian 6000 predicciones.

Ahora el modelo predice correctamente la fase de los estados desordenados, lo cual es un gran avance respecto al caso anterior. Encontramos el corte esperado entre ambas funciones a probabilidad del 50 % para cada etiqueta. Esto nos permite estimar la temperatura crítica como $T_c = 2{,}21$.

Una vez hecho este análisis es un buen momento para comparar todos los resultados nombrados del valor de la temperatura crítica en modelo de Ising:

Origen	$\frac{k_B T_c}{J}$
Artículo Onsager [1]	2.269
Simulación de Monte Carlo	2.28
Logistic Regression	2.15
DNN	2.21

A pesar de que la simulación de Monte Carlo ha sido más exacta, los métodos de Machine Learning se han aproximado bastante bien a la temperatura crítica (Aunque *Logistic Regression* no parece muy fiable). Un tratamiento más concienzudo o incluso un modelo más complejo podrían mejorar aún más las estimaciones, consiguiendo refinar los resultados y ganar exactitud.

4.4. *7-state clock model*: CNN

Finalmente, he implementado una CNN para detectar las 3 fases del *7-state clock model* estudiado anteriormente. Al ser un modelo de spines más complejo (y con vórtices en ubicaciones cambiantes), el uso de CNNs puede ser más efectivo al ser un modelo Machine Learning más avanzado. Describiré ahora la arquitecttura de la CNN empleada. Toma el input de 64×64 características

y le aplica dos fases de convolución+*pooling*. Las convoluciones se realizan fijándose en 5×5 píxeles y hace un *pooling* de 2×2 neuronas tras cada convolución. La primera capa convolucional es de profundidad 32 y la segunda, de profundidad 64, aprovechando que el *pooling* ha reducido el número de neuronas. El resultado de este doble proceso de convolución+*pooling* se aplana en un array 1D y se introduce en una capa tipo DNN de 64 neuronas y una salida *softmax*.

Durante el entrenamiento se emplea el optimizador Adam. Además, se emplean sólo datos lejanos a la zona crítica de las tres fases (separando dos particiones de dataset del resto correspondientes a dichos datos críticos), y se testea sobre un 30 % de los datos no críticos, reservando 20000 configuraciones del 70 % restante dedicado a entrenamiento a la validación del mismo. Se puede sacar un plot de la evolución de la precisión durante el entrenamiento sobre el set de entrenamiento y el de validación como el que se ve en la Figura 16.

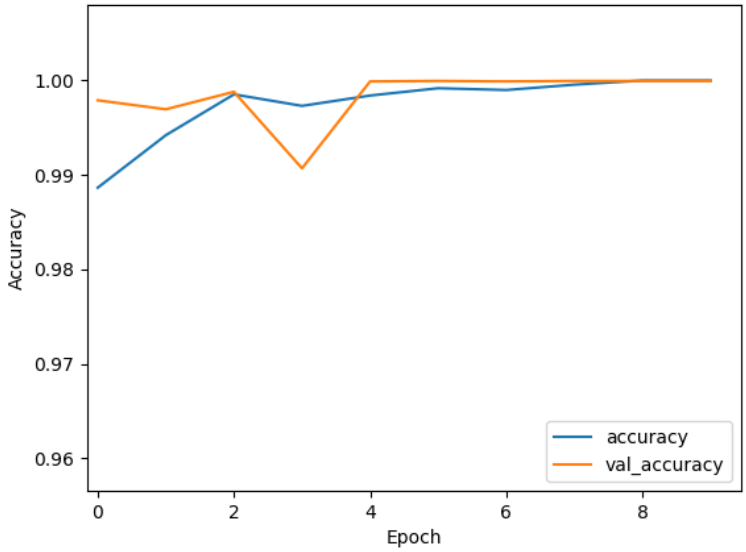


Figura 16: Evolución de la precisión en los sets de entrenamiento y validación respecto del *epoch* para la CNN. Se usa un tamaño de *batch* de 80 configuraciones

,

Se ve que las precisiones en ambos sets son muy altas desde el principio del entrenamiento. Por otro lado, las precisiones logradas en los datos del resto de particiones del dataset:

Partición	**Precisión**
Set de testeo	1.0
Set Crítico 1	0.81
Set Crítico 2	0.82

Donde la partición Crítica 1 tiene datos de la transición entre fase ordenada y fase KT y la Crítica 2, datos de la transición entre fase KT y desordenada. El programa que llevó a cabo el entrenamiento es MLClockCNN.py (Código en el Anexo D.7).He obtenido un sistema con una precisión en general más que aceptable.

Cumplido el objetivo de predecir correctamente las fases de configuraciones del *7-state clock model*, se puede pasar a realizar la simulación ya hecha en los otros apartados para estimar T_c (Figuras 13 y 15). Esta vez se aprovechará el otro dataset auxiliar ya descrito y empleando la técnica de promediar la predicción del modelo sobre los datos de cada temperatura. La simulación lleva a la Figura (17) y el código que la genera se llama Experimento626CNN.py (Código en el Anexo D.8).

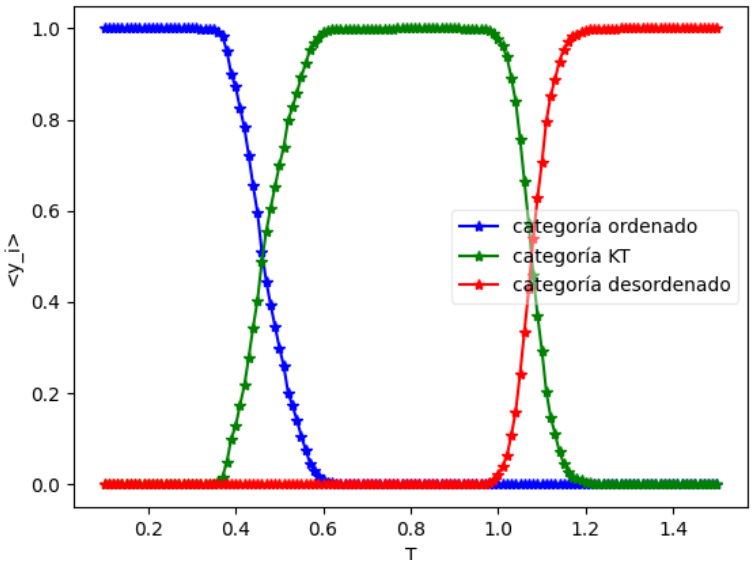


Figura 17: Evolución del promedio de la salida de la CNN según la temperatura. Por cada temperatura se promedian 2000 configuraciones

Con lo que podemos estimar las dos temperaturas críticas del modelo con sendos cortes de las funciones a $\langle y \rangle = 0{,}5$ y comparar con los resultados obtenidos mediante Monte Carlo y los de la literatura:

Origen	$\boldsymbol{T_{c1}}$	$\boldsymbol{T_{c2}}$
Referencia[3] (MC)	0.43	1.03
Simulación MC	0.44	1.04
CNN	0.46	1.08

De nuevo, hay más acuerdo entre la referencia y la simulación MC que lo que se obtiene con la CNN, pero los resultados están indudablemente bien encaminados. Es muy posible que se pueda mejorar el desempeño de la CNN en la detección de datos críticos, lo que debería traducirse en mejores resultados. Los resultados obtenidos con Machine Learning son una reproducción de lo obtenido en [11], aunque en ese artículo se emplean DNNs y $q = 6$.

5. Conclusiones

Este trabajo ha sido una comparativa entre métodos más clásicos y más novedosos para caracterizar transiciones de fase. Tras ver los resultados, vemos que existe acuerdo entre las conclusiones que alcanzan los métodos de Monte Carlo son las mismas que las de las DNNs en el modelo de Ising y que las de las CNNs en el *q-state clock model*.

Si bien es cierto que los resultados de MC han sido más exactos para ambos modelos de spines estudiados, esto no descarta la utilidad de las técnicas de Machine Learning. No sólo porque existe margen de mejora en las implementaciones que he hecho, sino también porque el potencial que han demostrado tener es valioso como herramienta. La eficacia como clasificadores de los modelos ha quedado bien clara en las implementaciones de los resultados.

El trabajo también ha servido para ver no solo las aplicaciones y resultados del uso de Machine Learning, sino también cómo se crea y optimiza un modelo,cómo hay que estar abiertos a sopesar modelos más complicados (Como ha sucedido ante al mal rendimiento de *Logistic Regression*) o enfrentarse a la posibilidad de requerir más datos (y a no poder generarlos).

Es importante recordar que los modelos de Machine Learning no tienen ningún conocimiento de como es el sístema físico, y a pesar de ello, el uso de sus fundamentos estadísticos y físicos le permiten hacer predicciones relevantes. Espero que en el desarrollo de esta memoria haya servido para acercar nuevos recursos como el Machine Learning a la ciencia.

6. Bibliografía

[1] Lars Onsager. "Crystal Statistics. I. A Two-Dimensional Model with an Order-Disorder Transition". En: *Phys. Rev.* 65 (3-4 feb. de 1944), págs. 117-149. DOI: 10.1103/PhysRev.65.117. URL: https://link.aps.org/doi/10.1103/PhysRev.65.117.

[2] G. Ortiz, E. Cobanera y Z. Nussinov. "Dualities and the phase diagram of the p-clock model". En: *Nuclear Physics B* 854.3 (2012), págs. 780-814. ISSN: 0550-3213. DOI: https://doi.org/10.1016/j.nuclphysb.2011.09.012. URL: http://www.sciencedirect.com/science/article/pii/S0550321311005219.

[3] Oscar A. Negrete y col. "Entropy and Mutability for the q-State Clock Model in Small Systems". En: *Entropy* 20 (12 2018). DOI: doi:10.3390/e20120933. URL: https://www.mdpi.com/1099-4300/20/12/933.

[4] Murty S. S. Challa y D. P. Landau. "Critical behavior of the six-state clock model in two dimensions". En: *Phys. Rev. B* 33 (1 ene. de 1986), págs. 437-443. DOI: 10.1103/PhysRevB.33.437. URL: https://link.aps.org/doi/10.1103/PhysRevB.33.437.

[5] Roman Krcmar, Andrej Gendiar y Tomotoshi Nishino. "Entanglement-Entropy Study of Phase Transitions in Six-State Clock Model". En: *Acta Physica Polonica A* 137 (mayo de 2020), págs. 598-600. DOI: 10.12693/APhysPolA.137.598.

[6] Jing Chen y col. "Phase Transition of the q -State Clock Model: Duality and Tensor Renormalization". En: *Chinese Physics Letters* 34 (mayo de 2017), pág. 050503. DOI: 10.1088/0256-307X/34/5/050503.

[7] Yusuke Tomita y Yutaka Okabe. "Probability-changing cluster algorithm for two-dimensional XY and clock models". En: *Phys. Rev. B* 65 (18 abr. de 2002), pág. 184405. DOI: 10.1103/PhysRevB.65.184405. URL: https://link.aps.org/doi/10.1103/PhysRevB.65.184405.

[8] Yukihiro Komura. *Improved CUDA programs for GPU computing of Swendsen–Wang multi-cluster spin flip algorithm: 2D and 3D Ising, Potts, and XY models*. 2019. DOI: 10.17632/7FNFBTSGRN.1. URL: https://data.mendeley.com/datasets/7fnfbtsgrn/1.

[9] Pankaj Mehta y col. "A high-bias, low-variance introduction to Machine Learning for physicists". En: *Physics Reports* 810 (2019). A high-bias, low-variance introduction to Machine Learning for physicists, págs. 1-124. ISSN: 0370-1573. DOI: https://doi.org/10.1016/j.physrep.2019.03.001. URL: http://www.sciencedirect.com/science/article/pii/S0370157319300766.

[10] Mourad Azhari y col. "Using Pyspark Environment for Solving a Big Data Problem: Searching for Supersymmetric Particles". En: *International Journal of Innovative Technology and Exploring Engineering* 9 (mayo de 2020). DOI: 10.35940/ijitee.G5308.059720.

[11] Kenta Shiina y col. "Machine-Learning Studies on Spin Models". En: *Scientific Reports* 10 (feb. de 2020), pág. 2177. DOI: 10.1038/s41598-020-58263-5.